Metody obliczeniowe w nauce i technice - sprawozdanie 9

Łukasz Jezapkowicz

14.05.2020

1 Tematem zadania będzie obliczanie metodami Monte Carlo całki funkcji x^2 oraz $\frac{1}{\sqrt{x}}$ w przedziale $(0, 1)$. Proszę dla obydwu funkcji:

1.1 Napisać funkcję liczącą całkę metodą "hit-and-miss". Czy będzie ona dobrze działać dla funkcji $\frac{1}{\sqrt{x}}$?

Metoda "hit-and-miss" pozwala obliczyć przybliżoną wartość całki. Polega ona na ograniczeniu obszaru całkowania przez prostokąt, którego lewym i prawym krańcem są brzegi przedziału, na którym całkujemy. Dolny oraz górny kraniec to odpowiednio wartość minimalna oraz maksymalna osiągana przez funkcję w przedziale całkowania. W ograniczonym obszarze losujemy punkty i sprawdzamy czy należą one do obszaru, który chcemy obliczyć (pola pod wykresem funkcji). Jeżeli wylosujemy n punktów, z których k znalazło się w naszym obszarze to nasza całka w przybliżeniu wynosi:

$$\int_a^b f(x)dx = \frac{k}{n}$$

Omówię teraz prosty program napisany w $C++$ realizujący to zadanie.

```
#define LEFT 0.0
#define RIGHT 1.0
#define UPPER 1.0 * 1e2
#define LOWER 1.0

/**
 * Function on which we want to calculate the Integral
 */
double f(double x) {
    return 1/sqrt(x);
}

/**
 * Random double number generator
 */
double random_double(double lower, double upper) {
    double f = (double) rand() / RAND_MAX;
    return lower + f * (upper-lower);
}

/**
 * Monte Carlo method
 */
double monte_carlo(int n) {

    int hit = 0;

    for (int i=0; i<n; i++) {
        double x = random_double(LEFT,RIGHT);
        double y = random_double(LOWER,UPPER);

        /* they are on the same side */
        if (f(x) * y >= 0) {
            if (((f(x) > 0) && (f(x) >= y)) || ((f(x) < 0) && (y >= f(x))))
                hit++;
        }
    }

    return 1.0 * hit / n;
}
```

Funkcja f odzwierciedla funkcję, po której całkujemy.
Funkcja *random_double* zwraca losową liczbę zmiennoprzecinkową z danego przedziału $[lower, upper]$.
Funkcja *monte_carlo* to właściwa część metody Monte Carlo losująca n punktów i sprawdzająca czy znajdują się one pod/nad wykresem (zależnie czy wartość funkcji jest dodatnia/ujemna). Zwraca przybliżoną wartość funkcji.

```
/**
 * Program calculating Integral based on Monte Carlo method
 */
int main(int argc, char *argv[]) {
    srand(time(NULL));

    /* number of hit-and-miss tries (10^n) */
    int n = atoi(argv[1]);

    for (int i=10; i<=pow(10,n); i*=10) {
        double result = monte_carlo(i);
        cout << setprecision(6) << "The integral is equal to " << result << " (" << i << " tries)\n";
    }

    return 0;
}
```

Ciało funkcji *main* jest bardzo proste. Na początku ustawiam losowy *seed*, a następnie wykonuję metodę Monte Carlo dla kolejnych potęg 10.
Spróbujmy teraz obliczyć wartości całki dla funkcji $\frac{1}{\sqrt{x}}$ oraz x^2 na przedziale $[0, 1]$. Obszar całkowania musimy ograniczyć pewnym prostokątem. Lewy i prawy kraniec to oczywiście 0 oraz 1, większy problem pojawia się dla poziomych krańców. Dla funkcji x^2 jest to jednak bardzo proste, ponieważ potrafimy znaleźć jej maksymalną i minimalną wartość na tym przedziale i są to odpowiednio 1 oraz 0. Przy takich ograniczeniach nasza funkcja daje następujące, przykładowe wyniki:

```
The integral is equal to 0.5 (10 tries)
The integral is equal to 0.38 (100 tries)
The integral is equal to 0.348 (1000 tries)
The integral is equal to 0.3297 (10000 tries)
The integral is equal to 0.33268 (100000 tries)
The integral is equal to 0.333458 (1000000 tries)
The integral is equal to 0.33324 (10000000 tries)
The integral is equal to 0.333323 (100000000 tries)
```

Analityczną wartością całki jest

$$\int_0^1 x^2 dx = \frac{1}{3}$$

więc widzimy, że wyniki, w miarę zwiększania liczebności pomiarów, zbiegają do tej liczby. Jednak widzimy, że dla 10^8 pomiarów błąd jest rzędu 10^5 co jest wynikiem słabo zadowalającym.
Funkcję $\frac{1}{\sqrt{x}}$ trudno jest nam ograniczyć z góry, ponieważ dąży ona w 0 do nieskończoności. Funkcja ta charakteryzuje się tym, że bardzo szybko maleje co skutkuje tym, że niemal zawsze wylosowany punkt nie będzie należał do naszego obszaru. Będzie tak, ponieważ przedział y, na których dokonujemy losowania będzie bardzo duży a tylko dla małej ilości x wartość funkcji jest duża. Skutkuje to tym, że nasza metoda jest bardzo nieskuteczna dla takiego rodzaju funkcji. Nawet jeśli ograniczymy nasz prostokąt z góry jakąś znacznie mniejszą liczbą niż nieskończoność (np. 100) to wyniki

nadal będą oderwane od rzeczywistości co pokazuje poniższy przykład (analityczna wartość całki to 2):

```
The integral is equal to 0 (10 tries)
The integral is equal to 0.03 (100 tries)
The integral is equal to 0.019 (1000 tries)
The integral is equal to 0.0165 (10000 tries)
The integral is equal to 0.02062 (100000 tries)
The integral is equal to 0.02009 (1000000 tries)
The integral is equal to 0.0199718 (10000000 tries)
The integral is equal to 0.0198964 (100000000 tries)
```

Wniosek: metoda "hit and miss" jest bardzo nieskuteczna dla małej ilości prób oraz niektórych funkcji, w których większość pola skupione jest na bardzo małym obszarze x'ów.

1.2 Policzyć całkę przy użyciu napisanej funkcji. Jak zmienia się błąd wraz ze wzrostem liczby podprzedziałów? Narysować wykres tej zależności przy pomocy Gnuplota. Przydatna będzie skala logarytmiczna.

W tym zadaniu użyjemy innej metody Monte Carlo. Polega ona na podzieleniu naszego obszaru całkowania na n podprzedziałów. W każdym z tych podprzedziałów wybieramy punkt środkowy będący średnią arytmetyczną kranców podprzedziału. Pomnożenie długości podprzedziału przez wartość funkcji w wybranym punkcie daje nam pole pewnego prostokąta. Przy pomocy podzielenia naszego obszaru na wiele takich prostokątów możemy przybliżyć wartość całki (całka to pole obszaru pod wykresem funkcji). W tym celu zmieniona została funkcja *monte_carlo*, która tym razem wygląda następująco:

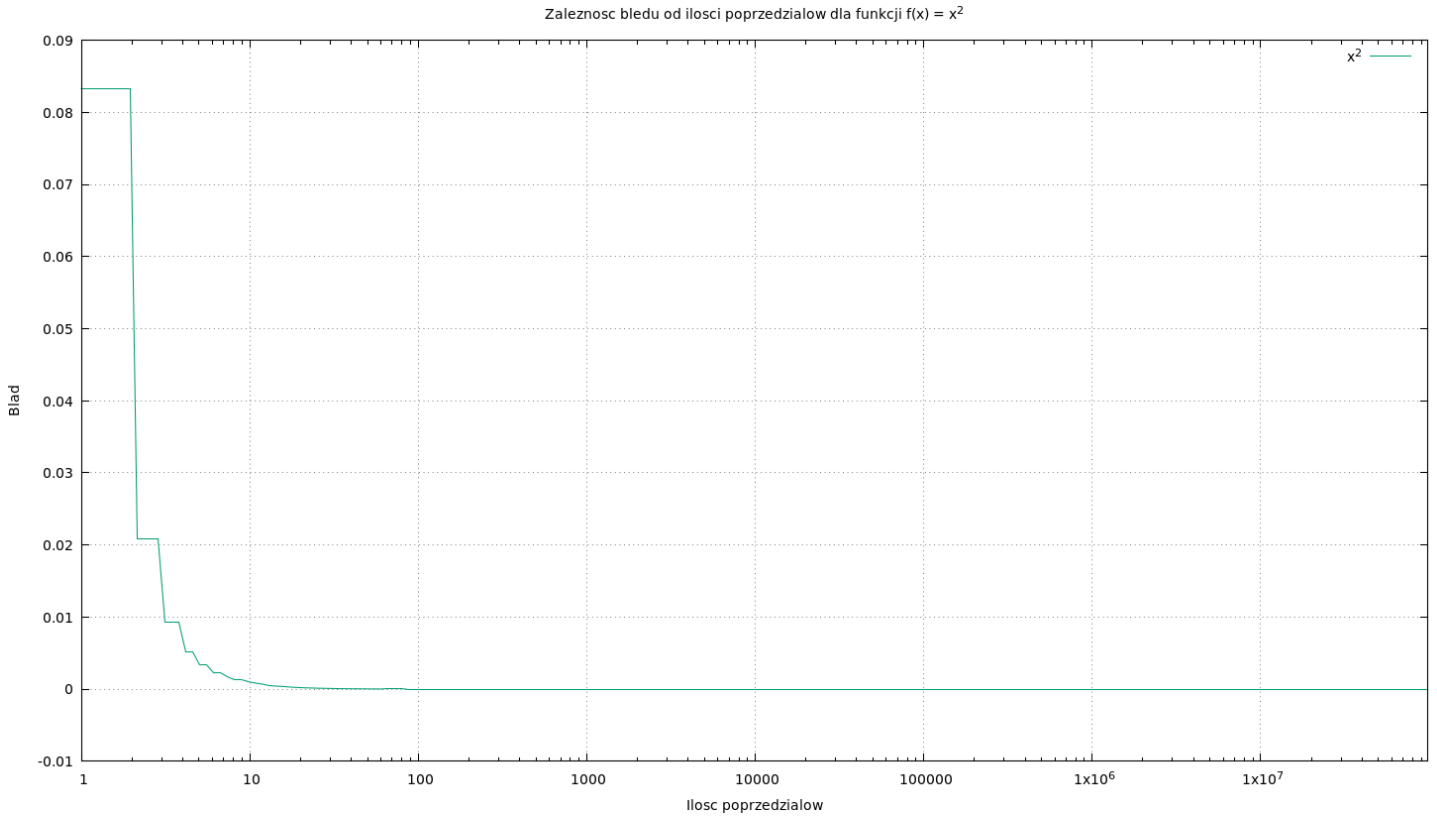
```c
/**
 * Monte Carlo method
 */
double monte_carlo(int n) {
    double len = 1.0 * (RIGHT-LEFT) / n;

    double mid = LEFT + len / 2.0;
    double sum = 0.0;

    for (int i=0; i<n; i++) {
        sum += 1.0 * len * f(mid);
        mid += 1.0 * len;
    }

    return sum;
}
```

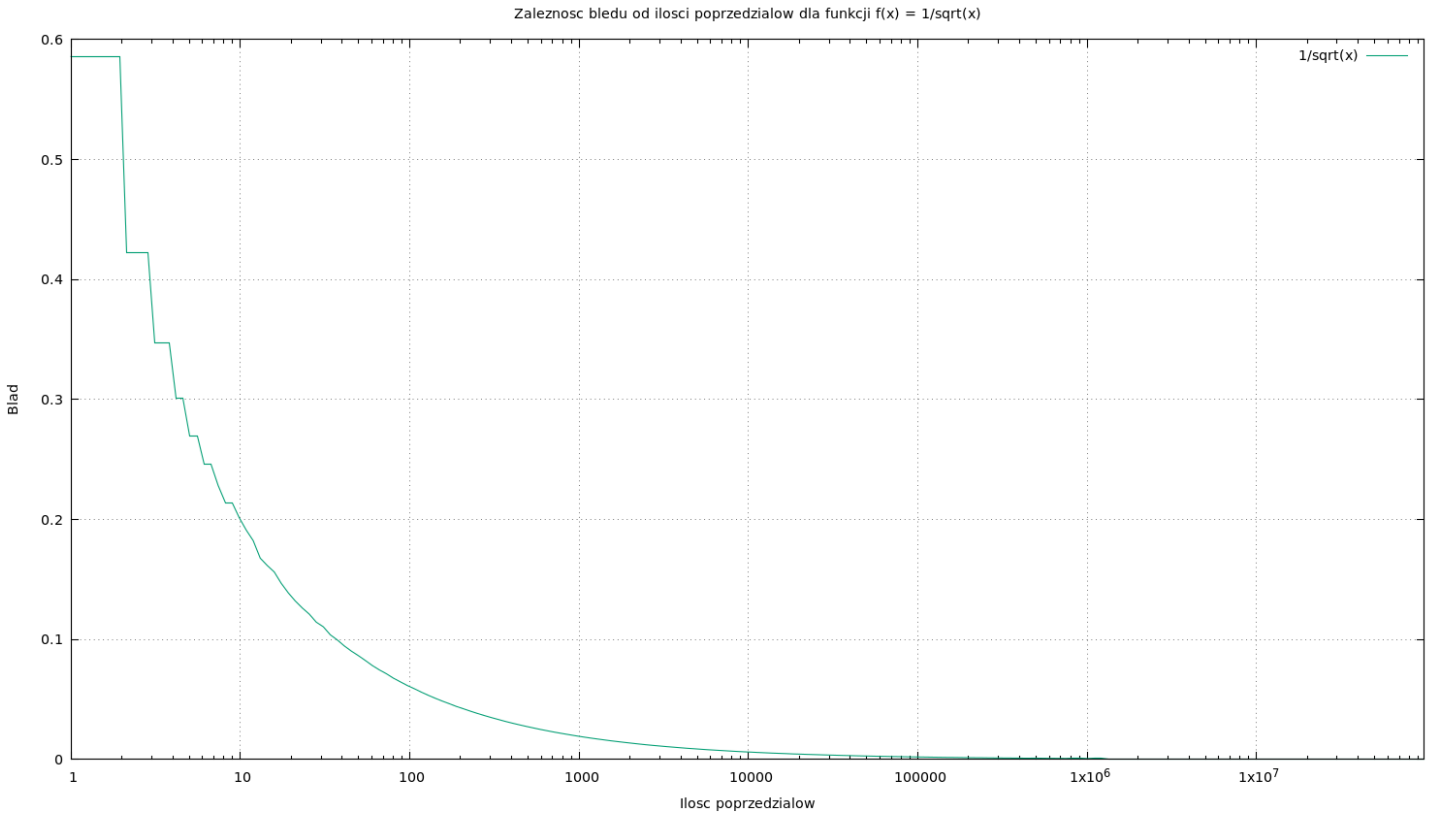
Przykładowe działanie dla funkcji odpowiednio x^2 oraz $\frac{1}{\sqrt{x}}$:

```
The integral is equal to 0.3325 (10 tries)
The integral is equal to 0.333325 (100 tries)
The integral is equal to 0.33333325 (1000 tries)
The integral is equal to 0.3333333325 (10000 tries)
The integral is equal to 0.3333333333 (100000 tries)
The integral is equal to 0.3333333333 (1000000 tries)
```

```
The integral is equal to 1.80892236 (10 tries)
The integral is equal to 1.939512219 (100 tries)
The integral is equal to 1.980871446 (1000 tries)
The integral is equal to 1.993951014 (10000 tries)
The integral is equal to 1.998087143 (100000 tries)
The integral is equal to 1.999395101 (1000000 tries)
The integral is equal to 1.999808714 (10000000 tries)
The integral is equal to 1.99993951 (100000000 tries)
```

Jak widać, ta metoda jest o wiele bardziej skuteczna niż metoda "hit and miss". Już dla $n = 10^6$ osiągnęliśmy zgodność na 10 miejscach po przecinku dla funkcji x^2. Dla funkcji $\frac{1}{\sqrt{x}}$ oraz $n = 10^8$ osiągnęliśmy błąd rzędu 10^5 co jest wynikiem o niebo lepszym niż błąd rzędu jedności jak w poprzedniej metodzie.
Teraz zaprezentuję wykres zmiany błędu obliczeń w zależności od ilości podprzedziałów dla obu funkcji.





Wniosek: Jak widać, w tej metodzie Monte Carlo wartość błędu bardzo szybko spada do wartości bliskich 0. Jest ona zatem o wiele skuteczniejsza niż metoda pokazana w zadaniu 1 oraz działa niemal dla każdej funkcji. Jest zatem zalecane używać tej metody zamiast metody "hit and miss".

1.3 Policzyć wartość całki korzystając z funkcji Monte Carlo z GSL.

Ponownie żeby skorzystać z funkcji Monte Carlo z biblioteki GSL potrzebujemy zmienić naszą funkcję *monte_carlo*. Poniżej widoczna zmieniona funkcja:

```c
/**
 * Monte Carlo method
 */
double monte_carlo(int n) {
    double res, err;
    double xl[1] = {LEFT};
    double xu[1] = {RIGHT};
    const gsl_rng_type *T;
    gsl_rng *r;
    gsl_monte_function G = {&f, 3, 0};
    size_t calls = n;
    gsl_rng_env_setup();
    T = gsl_rng_default;
    r = gsl_rng_alloc(T);

    gsl_monte_miser_state *s = gsl_monte_miser_alloc(1);
    gsl_monte_miser_integrate(&G,xl,xu,1,calls,r,s,&res,&err);
    gsl_monte_miser_free(s);

    gsl_rng_free(r);

    return res;
}
```

Poniżej widoczne wyniki działania dla odpowiednio x^2 oraz $\frac{1}{\sqrt{x}}$:

```
The integral is equal to 0.46018802 (10 tries)
The integral is equal to 0.2648590788 (100 tries)
The integral is equal to 0.3349282851 (1000 tries)
The integral is equal to 0.333352043 (10000 tries)
The integral is equal to 0.3333321874 (100000 tries)
The integral is equal to 0.3333333207 (1000000 tries)
The integral is equal to 0.3333333522 (10000000 tries)
The integral is equal to 0.3333333348 (100000000 tries)
```

```
The integral is equal to 1.460423464 (10 tries)
The integral is equal to 1.871827386 (100 tries)
The integral is equal to 2.082088786 (1000 tries)
The integral is equal to 2.002402786 (10000 tries)
The integral is equal to 1.997608741 (100000 tries)
The integral is equal to 2.000218514 (1000000 tries)
The integral is equal to 2.000007815 (10000000 tries)
The integral is equal to 1.999992163 (100000000 tries)
```

Wniosek: Co ciekawe, dla funkcji x^2 moja własna implementacja okazała się dawać lepsze wyniki niż funkcja wbudowana w bibliotece GSL. Biblioteka GSL pozwala jednak na obliczanie całek wielokrotnych co w samodzielnej implementacji jest o wiele trudniejszym zadaniem. Warto więc stosować tą bibliotekę do aproksymacji całek.